NAME

cproto – generate C function prototypes and convert function definitions

SYNOPSIS

cproto [*option* ...] [*file* ...]

DESCRIPTION

Cproto generates function prototypes for functions defined in the specified C source files to the standard output. The function definitions may be in the old style or ANSI C style. Optionally, **cproto** also outputs declarations for any variables defined in the file. If no *file* argument is given, **cproto** reads its input from the standard input.

By giving a command line option, **cproto** will also convert function definitions in the specified files from the old style to the ANSI C style. The original source files along with files specified by

```
#include "file"
```

directives appearing in the source code will be overwritten with the converted code. If no file names are given on the command line, then the program reads the source code from the standard input and outputs the converted source to the standard output.

If any comments appear in the parameter declarations for a function definition, such as in the example,

```
main (argc, argv)
int argc;       /* number of arguments */
char *argv[];   /* arguments */
{
 ...
}
```

then the converted function definition will have the form

```
int
main (
    int argc;       /* number of arguments */
    char *argv[];   /* arguments */
)
{
 ...
}
```

Otherwise, the converted function definition will look like

```
int
main (int argc, char *argv[])
{
 ...
}
```

Cproto can optionally convert function definitions from the ANSI style to the old style. In this mode, the program also converts function declarators and prototypes that appear outside function bodies. This is not a complete ANSI C to old C conversion. The program does not change anything within function bodies.

OPTIONS

–a Convert function definitions from the old style to the ANSI C style.

–e Output the keyword **extern** in front of every declaration having global scope.

–f*n* Set the style of function prototype where *n* is a number from 0 to 4. For example, consider the

function definition

```
main (argc, argv)
int argc;
char *argv[];
{
 ...
}
```

If the value is 0, then no prototypes are generated. When set to 1, the output is:

```
int main(/*int argc, char *argv[]*/);
```

For a value of 2, the output has the form:

```
int main(int /*argc*/, char */*argv*/[]);
```

The default value is 3. It produces the full function prototype:

```
int main(int argc, char *argv[]);
```

A value of 4 produces prototypes guarded by a macro:

```
int main P_((int argc, char *argv[]));
```

–m*name*

Set the name of the macro used to guard prototypes when option -f4 is selected. The default is "P_".

–d Omit the definition of the prototype macro named by the -m option.

–n Omit file name comments. The default is to output the source file name in a comment before generating the prototypes for that file.

–p Disable promotion of formal parameters in old style function definitions. By default, parameters of type **char** or **short** in old style function definitions are promoted to type **int** in the function prototype or converted ANSI C function definition. Parameters of type **float** get promoted to **double** as well.

–s By default, **cproto** only generates declarations for functions and variables having global scope. This option will output **static** declarations as well.

–t Convert function definitions from the ANSI C style to the traditional style.

–v Also output declarations for variables defined in the source.

–P*template*

–F*template*

–C*template*

Set the output format for generated prototypes, function definitions, and function definitions with parameter comments respectively. The format is specified by a template in the form

```
" int main ( a, b )"
```

but you may replace each space in this string with any number of whitespace characters. For example, the option

```
-F"int main(\n\ta,\n\tb\n\t)"
```

will produce

```
int main(
        int argc,
        char *argv[]
        )
```

–D*name[=value]*
This option is passed through to the preprocessor and is used to define symbols for use with conditionals such as *#ifdef*.

–U*name*
This option is passed through to the preprocessor and is used to remove any definitions of this symbol.

–I*directory*
This option is passed through to the preprocessor and is used to specify a directory to search for files that are referenced with *#include*.

–V Print version information.

ENVIRONMENT

The environment variable CPROTO is scanned for a list of options in the same format as the command line options.

BUGS

If an untagged struct, union or enum declaration appears in a generated function prototype or converted function definition, the content of the declaration between the braces is empty.

The program does not pipe the source files through the C preprocessor when it is converting function definitions. Instead, it tries to handle preprocessor directives and macros itself and can be confused by tricky macro expansions. The conversion also discards some comments in the function definition head.

When the program encounters an error, it usually outputs the not very descriptive message "syntax error".

Options that take string arguments only interpret the following character escape sequences:

\n	newline
\t	tab

AUTHOR

Chin Huang
cthuang@zerosan.UUCP
chin.huang@canrem.com

SEE ALSO

cc(1), cpp(1)